

SQL continued



Reading

☐ [RG] Sec 4.1 – 4.3, 15.3



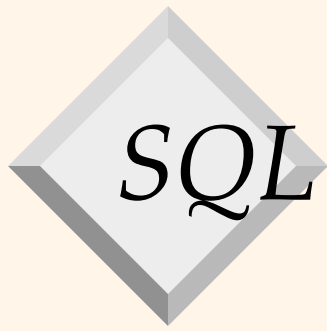
Reminders

- ☐ Make sure you're on Piazza and in CMS
- ☐ Course policies quiz – complete ASAP
- ☐ Office hours start this week
 - Schedule on CMS

Basic SQL Query

```
SELECT      [DISTINCT] target-list  
[FROM      relation-list]  
[WHERE     condition]
```

- SELECT * returns all attributes



SQL Query Example

```
SELECT S.sname  
FROM   Sailors S, Reserves R  
WHERE  S.sid=R.sid AND R.bid=101;
```



Find names of sailors who have reserved a red OR a blue boat

```
SELECT S.sname  
FROM Sailors S, Boats B, Reserves R  
WHERE R.sid=S.sid AND R.bid=B.bid  
      AND (B.color='red' OR B.color='blue');
```




Find names of sailors who have reserved a red OR a blue boat

```
SELECT S.sname
FROM Sailors S, Boats B, Reserves R
WHERE R.sid=S.sid AND R.bid=B.bid
      AND (B.color='red' OR B.color='blue');
```

- **UNION**: Compute union of any two **union-compatible** sets of tuples

```
SELECT S.sname
FROM Sailors S, Boats B, Reserves R
WHERE R.sid=S.sid AND R.bid=B.bid
      AND B.color='red'
UNION
SELECT S.sname
FROM Sailors S, Boats B, Reserves R
WHERE R.sid=S.sid AND R.bid=B.bid
      AND B.color='blue';
```



UNION ALL

☐ To keep duplicates (if you so desire) use
UNION ALL

```
SELECT S.sname  
FROM Sailors S, Boats B, Reserves R WHERE R.sid=S.sid  
AND R.bid=B.bid  
AND B.color='red'  
UNION ALL  
SELECT S.sname  
FROM Sailors S, Boats B, Reserves R WHERE R.sid=S.sid  
AND R.bid=B.bid  
AND B.color='blue';
```




*Find names of sailors who have reserved a
red and a blue boat*



Can we just do this?

```
SELECT S.sname  
FROM Sailors S, Boats B, Reserves R  
WHERE R.sid=S.sid AND R.bid=B.bid  
      AND (B.color='red' AND B.color='blue');
```



Find names of sailors who have reserved a red and a blue boat

```
SELECT S.sname
FROM Sailors S, Boats B1, Boats B2, Reserves R1, Reserves R2
WHERE S.sid=R1.sid AND R1.bid=B1.bid
      AND S.sid=R2.sid AND R2.bid=B2.bid
      AND (B1.color='red' AND B2.color='blue');
```




*Can we use the **INTERSECT** operator?*

- ▶ **UNION** worked for us before, how about these two queries?
- ▶ Also: try it in MySQL and see what happens!

```
SELECT S.sname
FROM Sailors S, Boats B1, Boats B2, Reserves
R1, Reserves R2
WHERE S.sid=R1.sid AND R1.bid=B1.bid
AND S.sid=R2.sid AND R2.bid=B2.bid
AND (B1.color='red' AND B2.color='blue');
```

```
SELECT S.sname
FROM Sailors S, Boats B, Reserves R
WHERE R.sid=S.sid AND R.bid=B.bid
AND B.color='red'
INTERSECT
SELECT S.sname
FROM Sailors S, Boats B, Reserves R
WHERE R.sid=S.sid AND R.bid=B.bid
AND B.color='blue';
```



An easier query with INTERSECT

```
SELECT S.sid
FROM Sailors S, Boats B, Reserves R WHERE R.sid=S.sid AND
R.bid=B.bid
AND B.color='red'
INTERSECT
SELECT S.sid
FROM Sailors S, Boats B, Reserves R WHERE R.sid=S.sid AND
R.bid=B.bid
AND B.color='blue';
```

EXCEPT

- ▶ Also available: **EXCEPT**
 - What do you think it does?

```
SELECT S.sid  
FROM Sailors S, Boats B, Reserves R  
WHERE R.sid=S.sid AND R.bid=B.bid  
AND B.color='red'  
EXCEPT  
SELECT S.sid  
FROM Sailors S, Boats B, Reserves R  
WHERE R.sid=S.sid AND R.bid=B.bid  
AND B.color='blue';
```



Nested Queries

Find names of sailors who've reserved boat 101:

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN (SELECT R.sid
                FROM Reserves R
                WHERE R.bid=101);
```

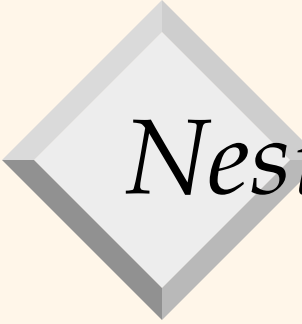
- ❑ Powerful SQL feature: WHERE clause can contain an SQL query
 - (Actually, so can other clauses e.g. FROM)
- ❑ To find sailors who have *not* reserved 101, use NOT IN

Nested Queries (with Correlation)

Find names of sailors who have reserved boat #101:

```
SELECT S.sname  
FROM Sailors S  
WHERE EXISTS (SELECT *  
               FROM Reserves R  
               WHERE R.bid=101 AND S.sid=R.sid);
```





Nested Queries (with Correlation)

*Find names of sailors who have **not** reserved boat #101:*

```
SELECT  S.sname
FROM    Sailors S
WHERE   NOT EXISTS (SELECT *
                      FROM Reserves R
                      WHERE R.bid=101 AND S.sid=R.sid);
```



Now for a harder puzzle

Find sailors who've reserved all boats



Find sailors who've reserved all boats

```
SELECT S.sname
FROM   Sailors S
WHERE  NOT EXISTS ((SELECT B.bid
                     FROM   Boats B)
                  EXCEPT
                  (SELECT R.bid
                     FROM   Reserves R
                     WHERE  R.sid=S.sid));
```



The same thing without Except!

Find sailors who've reserved all boats.

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (SELECT B.bid
FROM Boats B
WHERE NOT EXISTS (SELECT R.bid
FROM Reserves R
WHERE R.bid=B.bid
AND R.sid=S.sid));
```

Sailors S such that ...
there is no boat B without ...
a Reserves tuple showing S reserved B



More on Set-Comparison Operators

- *op* ANY, *op* ALL
 - *op* can be $>$, $<$, $=$, \geq , \leq , \neq
- Find sailors whose rating is greater than that of all sailors called Bob:

```
SELECT *  
FROM Sailors S  
WHERE S.rating  $>$  ALL (SELECT S2.rating  
                      FROM Sailors S2  
                      WHERE S2.sname='Bob');
```

Aggregate Operators

COUNT (*)
COUNT ([DISTINCT] A)
SUM ([DISTINCT] A)
AVG ([DISTINCT] A)
MAX (A)
MIN (A)

single column

```
SELECT COUNT(*)  
FROM Sailors S;
```

```
SELECT AVG (S.age)  
FROM Sailors S  
WHERE S.rating=9;
```

```
SELECT COUNT(DISTINCT S.rating)  
FROM Sailors S  
WHERE S.sname='Bob';
```



Find name and age of the oldest sailor(s)

- ▶ First query is illegal
 - Although MySQL happily allows it...
 - Let's experiment!

```
SELECT S.sname, MAX(S.age)
FROM Sailors S;
```

```
SELECT S.sname, S.age
FROM Sailors S
WHERE S.age =
      (SELECT MAX(S2.age)
       FROM Sailors S2);
```

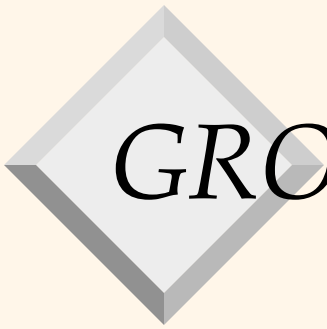


Aggregate Operators

- Sometimes, we want to apply aggregates over *groups* of tuples.
- Consider: *Find the age of the youngest sailor for each rating level.*
 - If rating values go from 1 to 10; we can write 10 queries that look like this:

For $i = 1, 2, \dots, 10$:

```
SELECT MIN (S.age)
FROM Sailors S
WHERE S.rating =  $i$ 
```

GROUP BY

```
SELECT    S.rating, MIN(S.Age)
FROM      Sailors S
GROUP BY  S.rating;
```

☐ Evaluation process:

- Compute result of SELECT-FROM(-WHERE)
- Partition based on GROUP BY criteria
- Output one answer for each group

Age of youngest sailor for each rating

rating	age
2	45.0
6	33.0
4	55.5
4	25.5
1	35.0
2	35.0
1	16.0
3	35.0
5	25.5
5	63.5
5	25.5



rating	age
2	45.0
2	35.0
6	33.0
4	55.5
4	25.5
1	35.0
1	16.0
3	35.0
5	25.5
5	63.5
5	25.5



rating	minage
4	25.5
2	35.0
5	25.5
6	33.0
3	35.0
1	16.0



Caution! Illegal GROUP BY query

```
SELECT S.rating, S.sname, MIN(S.age) AS minage  
FROM Sailors S  
GROUP BY S.rating;
```

Usual trick: try in Postgres and MySQL...



Queries With GROUP BY

SELECT	[DISTINCT] <i>target-list</i>
FROM	<i>relation-list</i>
WHERE	<i>qualification</i>
GROUP BY	<i>grouping-list</i>

- ❑ *target-list* contains attribute names and terms with aggregates, e.g., MIN (P.age)
- ❑ Attributes in *target-list* that are not arguments to an aggregate must be in *grouping-list*
 - Intuition: Each answer tuple corresponds to a group, and these attributes must have a single value per group

*Age of youngest adult sailor for ratings w/ at least two **such** sailors*

```
SELECT S.rating, MIN(s.age) AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT(*) > 1;
```

Answer relation:

rating	minage
4	25.5
2	35.0
5	25.5

<u>sid</u>	sname	rating	age
22	dustin	2	45.0
29	brutus	6	33.0
31	lubber	4	55.5
32	andy	4	25.5
58	rusty	1	35.0
64	horatio	2	35.0
71	zorba	1	16.0
74	horatio	3	35.0
85	art	5	25.5
95	bob	5	63.5
96	frodo	5	25.5

Age of youngest adult sailor for ratings w/ at least two such sailors

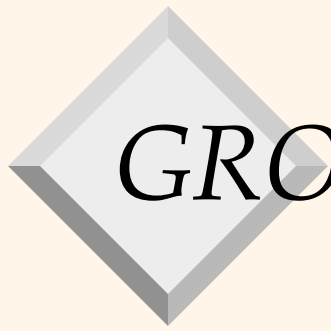
rating	age
2	45.0
6	33.0
4	55.5
4	25.5
1	35.0
2	35.0
1	16.0
3	35.0
5	25.5
5	63.5
5	25.5



rating	age
2	45.0
2	35.0
6	33.0
4	55.5
4	25.5
1	35.0
3	35.0
5	25.5
5	63.5
5	25.5



rating	minage
4	25.5
2	35.0
5	25.5



GROUP BY and HAVING

SELECT	[DISTINCT] <i>target-list</i>
FROM	<i>relation-list</i>
WHERE	<i>qualification</i>
GROUP BY	<i>grouping-list</i>
HAVING	<i>group-qualification</i>



HAVING clause

- ❑ Expression in HAVING clause **must have single value per group**
- ❑ Output one answer tuple per qualifying group

Age of youngest adult sailors for ratings with at least 2 sailors over 18, but all under 60

rating	age
2	45.0
6	33.0
4	55.5
4	25.5
1	35.0
2	35.0
1	16.0
3	35.0
5	25.5
5	63.5
5	25.5



rating	age
2	45.0
2	35.0
6	33.0
4	55.5
4	25.5
1	35.0
3	35.0
5	25.5
5	63.5
5	25.5



rating	minage
4	25.5
2	35.0

HAVING COUNT(*) > 1 AND EVERY (S.age <=60)

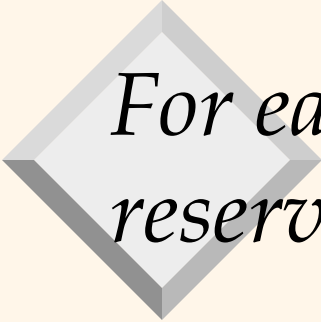


HAVING fun...

```
SELECT S.sname  
FROM Sailors S  
GROUP BY S.sname  
HAVING S.rating=9;
```

```
SELECT S.sid  
FROM Sailors S  
GROUP BY S.sid  
HAVING S.rating = 9;
```


- ❑ Second query "should" be safe because S.sid is a primary key!
- ❑ But illegal according to standard
 - S.rating must be mentioned in GROUP BY clause



*For each blue boat, find number of
reservations for this boat*

```
SELECT B.bid, COUNT(*) AS rcount  
FROM Boats B, Reserves R  
WHERE R.bid=B.bid AND B.color='blue'  
GROUP BY B.bid;
```

❑ Grouping over join



Find ratings for which average age is minimum over all ratings

Aggregate operations cannot be nested. **WRONG:**

```
SELECT S.rating
FROM Sailors S
WHERE AVG(S.age) = (SELECT MIN(AVG(S2.age)) FROM Sailors S2);
```

Correct(ish) solution:

```
SELECT Temp.rating
FROM (SELECT S.rating , AVG (S.age) AS avgage
      FROM Sailors S
      GROUP BY S.rating) AS Temp
WHERE Temp.avgage = (SELECT MIN(Temp.avgage) FROM Temp);
```



Or more precisely, if you actually want it to run....

```
SELECT Temp.rating
FROM (SELECT S.rating , AVG (S.age) AS avgage
      FROM Sailors S
      GROUP BY S.rating) AS Temp
WHERE Temp.avgage = (SELECT MIN(Temp2.avgage)
                    FROM
                      (SELECT S2.rating, AVG (S2.age) AS avgage
                       FROM Sailors S2
                       GROUP BY S2.rating) AS Temp2);
```